

"IBM continues to build the past, Microsoft tries to hold us in present, and Oracle continues to drive us into the future..."

*Rich Niemiec,
president of the International Oracle Users Group (IOUG)*



PL/SQL

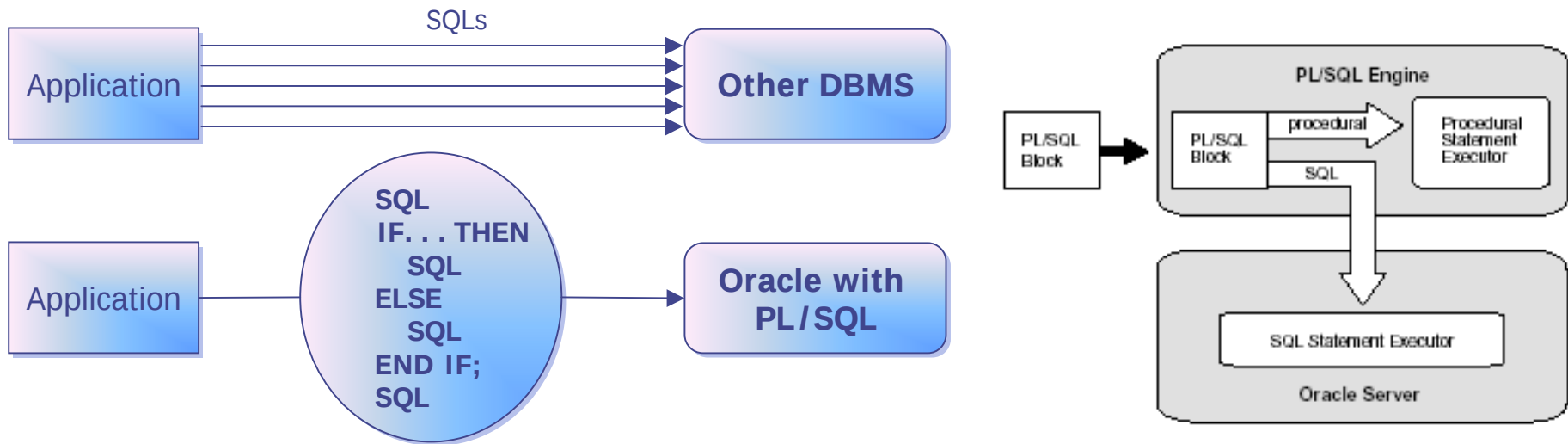
'uiael keg cipe`l x"c

leokhg@012.net.il

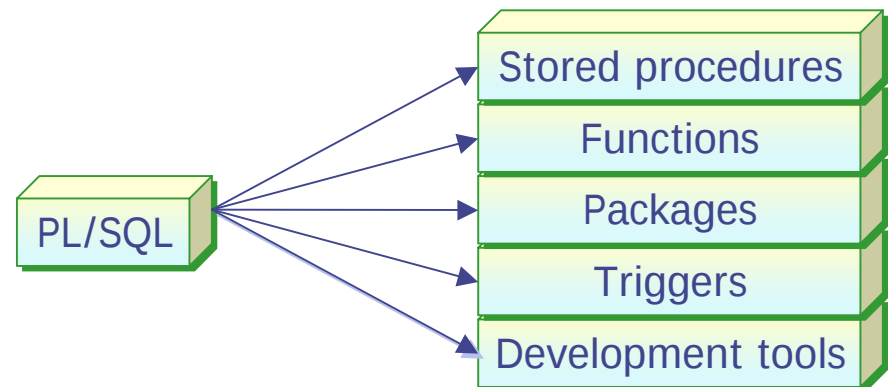
www.geocities.com/leokhg

PL/SQL

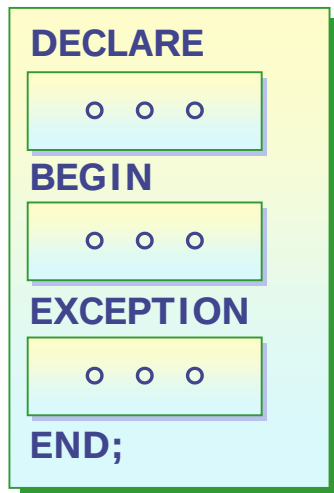
- PL/SQL is an extension to SQL with design features of programming languages.
- Data manipulation and query statements of SQL are included within procedural units of code.



Procedural Language/SQL (PL/SQL) is Oracle Corporation's procedural language extension to SQL, the standard data access language for relational databases. PL/SQL offers modern software engineering features such as data encapsulation, exception handling, information hiding, and object orientation, and so brings state-of-the-art programming to the Oracle Server and Toolset.



PL/SQL Block Structure

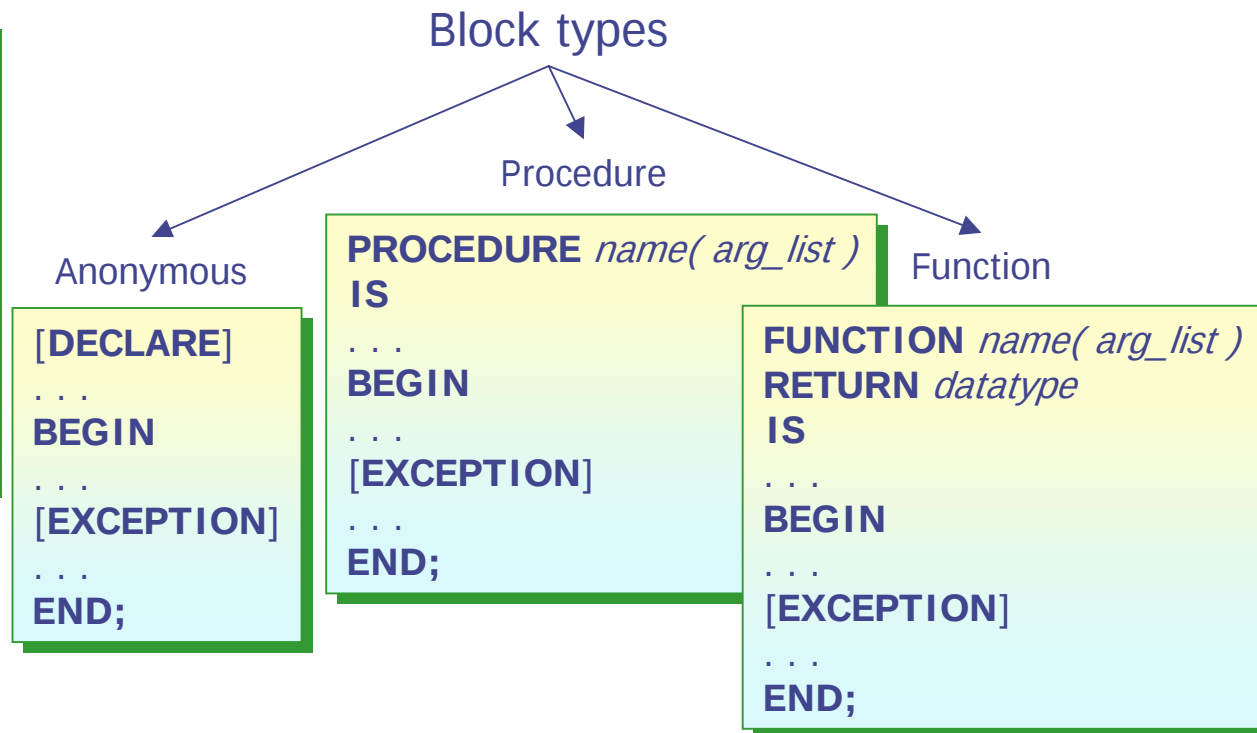


- DECLARE**
Variables, constants, cursors, user-defined exceptions
referenced in the executable and declarative sections
- **BEGIN**
 - SQL statements
 - PL/SQL statements
 - **EXCEPTION**
Actions to perform when errors and abnormal conditions occur in the executable section
 - **END;**

```

DECLARE
  v_variable VARCHAR2( 5 );
BEGIN
  SELECT column_name
    INTO v_variable
  FROM table_name;
EXCEPTION
  WHEN exception_name THEN
    ...
END;

```



Declaring PL/SQL Variables

Syntax:

identifier [CONSTANT] ***datatype*** [NOT NULL] [:= | DEFAULT ***expr***] ;

Examples:

```
DECLARE
  v_hiredate DATE;
  v_deptno NUMBER (2) NOT NULL :=10;
  v_location VARCHAR2(13) := 'Atlanta';
  c_comm CONSTANT  NUMBER :=1400;
```

Base scalar datatypes

- VARCHAR2(max_length)
- NUMBER[(p, s)]
- DATE
- CHAR[(length)]
- LONG
- BOOLEAN
- BINARY_INTEGER
- PLS_INTEGER

In the syntax:

<i>identifier</i>	is the name of the variable (should not have the same name as name of table columns used in the block)
CONSTANT	constrains the variable so that its value cannot change (Constants must be initialized).
<i>datatype</i>	is a scalar, composite, reference or LOB data type
NOT NULL	constrains the variable so that it must contain a value (NOT NULL variables must be initialized).
<i>expr</i>	is any PL/SQL expression that can be a literal, another variable or an expression involving operators and functions

```
v_job      VARCHAR2(9);
v_count    BINARY_INTEGER := 0;
v_total_sal NUMBER( 9, 2 ) := 0;
v_orderdate DATE := SYSDATE + 7;
```

```
c_tax_rate  CONSTANT NUMBER( 3, 2 ) := 8.25;
v_valid     BOOLEAN NOT NULL := TRUE;
v_ename     emp.ename%TYPE;
v_balance   v_total_sal%TYPE;
```

PL/SQL Variables

Assignment: *identifier := expr ;*

Where *expr ::= { constant / variable / function call / expression } , but not a database column !*

Composite datatypes

PL/SQL RECORD

1723644	TRUE	23-DEC-98	ATLANTA	
---------	------	-----------	---------	--

PL/SQL TABLE

1	SMITH
2	JONES
3	NANCY
4	TIM

↑ ↑
 BINARY_INTEGER VARCHAR2

LOB datatypes

CLOB

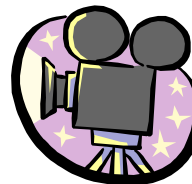


NCLOB

BLOB



BFILE



The %TYPE prefix

- *tbl.column* %TYPE
- *previously_declared_var* %TYPE

```

DECLARE
  v_sum  NUMBER( 9, 2 );
  v_id   emp.empno%TYPE;
  v_total v.sum%TYPE;
BEGIN
  . . .
  
```

Comments

- Single-line comments
- Multi-line comments

a := 0; -- this is a comment
/ All this text*
*is comments */*

SQL Function in PL/SQL

➤ Available in procedural statements:

- Single-row string
- Single-row number
- Datatype conversion
- Date



Same as in SQL

➤ Not available in procedural statements:

- **DECODE**
- **Group function**

```

DECLARE
  v_hour   NUMBER(2);           -- for number of hours
  v_min    NUMBER(2);           -- for number of minutes
  v_SysD   DATE;               -- for system date
  v_res    VARCHAR2(16);       -- for result

BEGIN
  SELECT SYSDATE INTO v_SysD FROM DUAL;           -- read system date
  v_hour  := TO_NUMBER( TO_CHAR( v_SysD, 'HH' ) ); -- calculate number of hours
  v_min   := TO_NUMBER( TO_CHAR( v_SysD, 'MI' ) ); -- calculate number of minutes
  v_res   := TRIM(TO_CHAR( v_hour, '00' )) || ':' ||
             TRIM(TO_CHAR( v_min, '00' )) ||
             ' ' || TO_CHAR( v_SysD, 'AM' );         -- compose result
  DBMS_OUTPUT.PUT_LINE( v_res );                     -- show result

END;
```

Nested Blocks

```

. . .
v_x NUMBER(5,2);
BEGIN
    . . .
    DECLARE
        v_x VARCHAR2( 20 );
        v_y DATE;
    BEGIN
        . . .
    END;
    . . .
END;

```

- Statements can be nested wherever an executable statement is allowed
- A nested block becomes a statement
- An exception section can contain nested blocks
- A block can look up to the enclosing block
- A block cannot look down to enclosed blocks

```

DECLARE
    v_sal    NUMBER(7,2) := 60000;
    v_comm   NUMBER(7,2) := V_SAL * 0.20;
    v_msg    VARCHAR2(255) := ' eligible for commision';
BEGIN . . .
    DECLARE
        v_sal    NUMBER(7,2) := 50000;
        v_comm   NUMBER(7,2) := 0;
        v_total  NUMBER(8,2) := v_sal + v_comm;
    BEGIN . . .
        v_msg := 'CLERK is not' || v_msg;
    END;
    v_msg := 'SALESMAN is' || v_msg;
END;

```

Logical flow control statement

```
IF condition THEN  
    statements ;  
[ ELSEIF condition THEN  
    statements ; ]  
[ ELSE  
    statements ; ]  
END IF;
```

condition
{ **TRUE** | **FALSE** | **NULL** }

```
SET SERVEROUTPUT ON  -- SQL *Plus command  
DECLARE  
    v_cnt NUMBER(5);  
    v_tot v_cnt%TYPE;  
BEGIN  
    SELECT COUNT( comm ), COUNT( * )  
    INTO v_cnt, v_tot FROM emp;  
    IF v_cnt = 0 THEN  
        DBMS_OUTPUT.PUT_LINE( 'Nobody' );  
    ELSIF v_cnt = 1 THEN  
        DBMS_OUTPUT.PUT_LINE( 'Only one' );  
    ELSIF v_cnt > 1 AND v_cnt <= v_tot / 2 THEN  
        DBMS_OUTPUT.PUT_LINE( 'Less than half' );  
    ELSIF v_cnt > v_tot / 2 AND v_cnt < v_tot THEN  
        DBMS_OUTPUT.PUT_LINE( 'More than half' );  
    ELSE  
        DBMS_OUTPUT.PUT_LINE( 'Everybody' );  
    END IF;  
END;
```

Loop statements

Basic loop	LOOP <i>statement1</i> ; ... EXIT [WHEN <i>condition</i>] END LOOP ;		<pre> DECLARE v_counter NUMBER(2) := 1; BEGIN LOOP INSERT INTO item(order_id, item_id) VALUES(622, v_counter); v_counter := v_counter + 1; EXIT WHEN v_counter > 10; END LOOP; END; ----- BEGIN FOR i IN 1..10 LOOP INSERT INTO item(order_id, item_id) VALUES(622, i); END LOOP; END; ----- DECLARE v_counter NUMBER(2) := 1; BEGIN WHILE v_counter <= 10 LOOP INSERT INTO item(order_id, item_id) VALUES (622, v_counter); v_counter := v_counter + 1; END LOOP; END; </pre>
FOR loop	FOR <i>counter</i> IN [REVERSE] <i>Lbound .. Ubound</i> LOOP <i>statement1</i> ; <i>statement2</i> ; ... END LOOP ;		
WHILE loop	WHILE <i>condition</i> LOOP <i>statement1</i> ; <i>statement2</i> ; ... END LOOP;		
Cursor FOR loop	?		

SQL statements in PL/SQL

```

SELECT select_list
INTO { variable_name [, variable_name ]...
        | record_name }
FROM table
WHERE condition

```

```

DECLARE
    v_deptno    NUMBER(2);
    v_loc       VARCHAR2(15);
BEGIN
    SELECT deptno, loc
    INTO v_deptno, v_loc
    FROM dept
    WHERE dname = 'SALES'
    . . .
END;

```

Query MUST return one and Only one row !

```

DECLARE
    v_sal_increase emp.sal%TYPE := 1000;
    v_new_empno emp.empno%TYPE;
    v_empty_dept dept.deptno%TYPE := 40;
BEGIN
    UPDATE EMP SET SAL = SAL + v_sal_increase
    WHERE JOB = 'ANALYST';

    SELECT NVL( MAX( EMPNO ) ) + 1 INTO v_new_empno
    FROM EMP;

    INSERT INTO EMP( EMPNO, ENAME, JOB, SAL, DEPTNO )
    VALUES( v_new_empno, 'ELLIS', 'CLERK', 1100, 20 );

    DELETE FROM DEPT
    WHERE DEPTNO = v_empty_dept;
END;

```

Sequences

```
CREATE SEQUENCE seq_name  
[ INCREMENT BY integer ]  
[ START WITH integer ]  
[ MAXVALUE integer | NOMAXVALUE ]  
[ MINVALUE integer | NOMINVALUE ]  
[ CYCLE | NOCYCLE ]  
[ CACHE | NOCACHE ];
```

```
ALTER SEQUENCE seq_name  
[ INCREMENT BY integer ]  
[ MAXVALUE integer | NOMAXVALUE ]  
[ MINVALUE integer | NOMINVALUE ]  
[ CYCLE | NOCYCLE ]  
[ CACHE | NOCACHE ];
```

```
DROP SEQUENCE seq_name;
```

NEXTVAL

Returns the next available sequence value

CURRVAL

Obtainss the current sequence value

- Automatically generates unique numbers
- Is shared object
- Is typically used to create primary key value
- Replaces application code
- Speeds up the efficiency of accessing sequence values when cached in memory



Sequences (cont.)

```
SQL> SELECT * FROM DEPT;
```

DEPTNO	DNAME	LOC
-----	-----	-----
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

```
SQL> CREATE SEQUENCE DEPT_SEQ  
2 INCREMENT BY 10  
3 START WITH 50  
4 MINVALUE 10  
5 MAXVALUE 99;
```

Sequence created.

```
SQL> INSERT INTO DEPT VALUES( DEPT_SEQ.NEXTVAL, 'SOFTWARE', 'HUSTON' );
```

1 row created.

```
SQL> INSERT INTO DEPT VALUES( DEPT_SEQ.NEXTVAL, 'HARDWARE', 'CHICAGO' );
```

1 row created.

```
SQL> SELECT * FROM DEPT;
```

DEPTNO	DNAME	LOC
-----	-----	-----
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	SOFTWARE	SACRAMENTO
60	HARDWARE	CHICAGO



USER_SEQUENCES

Triggers

A trigger defines an action the database should take when some of database-related events occurs

```
CREATE [ OR REPLACE ] TRIGGER trig_name
{ BEFORE | AFTER | INSTEAD OF }
{ DELETE | INSERT | UPDATE [ OF column [, column] ... ] }
[ OR { DELETE | INSERT | UPDATE [ OF column [, column] ... ] } ] ...
ON { tab_name | view_name }
[ REFERENCING { OLD old | NEW new } ]
[ FOR EACH ROW ]
[ WHEN ( conditions ) ]
plsql_block
```

```
ALTER TRIGGER trig_name { ENABLE | DISABLE | COMPILE [ DEBUGG ] }
```

```
ALTER TABLE tab_name
{ ENABLE | DISABLE } ALL TRIGGERS
```

```
DROP TRIGGER trig_name
```

Using trigger for primary key generation

```
SQL> CREATE OR REPLACE TRIGGER trig_new_emp
 2 BEFORE INSERT ON DEPT
 3 FOR EACH ROW
 4 DECLARE
 5     v_dno NUMBER(2);
 6 BEGIN
 7     SELECT dept_seq.NEXTVAL
 8     INTO v_dno FROM dual;
 9     :NEW.DEPTNO := v_dno;
10 END;
11 /
```

Trigger created.

```
SQL> INSERT INTO DEPT( DNAME, LOC )
 2 VALUES( 'SOFTWARE', 'SACRAMENTO' );
```

1 row created.

```
SQL> INSERT INTO DEPT( DNAME, LOC )
 2 VALUES( 'HARDWARE', 'CHICAGO' );
```

1 row created.

```
SQL> SELECT * FROM DEPT;
```

DEPTNO	DNAME	LOC

10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

```
SQL> SELECT * FROM DEPT;
```

DEPTNO	DNAME	LOC

10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON
50	SOFTWARE	SACRAMENTO
60	HARDWARE	CHICAGO

Using trigger for monitoring

```
SQL> CONNECT system/manager
Connected.
```

```
SQL> DROP TABLE secret1;
```

Table dropped.

```
SQL> CREATE TABLE secret1
  2  (
  3      vdate DATE,
  4      empno NUMBER(4),
  5      old_sal NUMBER(7,2),
  6      new_sal NUMBER(8,2),
  7      user_id NUMBER(5)
  8  );
```

Table created.

```
SQL> GRANT INSERT ON secret1
  2  TO scott;
```

Grant succeeded.

1

```
SQL> CONNECT scott/tiger
Connected.
```

```
SQL> CREATE OR REPLACE TRIGGER sal_secret1
  2  AFTER UPDATE OF sal
  3  ON emp
  4  FOR EACH ROW
  5  WHEN ( NEW.sal / OLD.sal > 1.1 )
  6  BEGIN
  7      INSERT INTO system.secret1
  8      VALUES( SYSDATE, :OLD.empno,
                :OLD.sal, :NEW.sal, UID );
  9  END;
10  /
```

Trigger created.

```
SQL> UPDATE emp
  2  SET sal = sal + 200;
```

14 rows updated.

```
SQL> COMMIT;
```

Commit complete.

2

Using trigger for monitoring (cont.)

```
SQL> CONNECT system/manager
Connected.
SQL> SELECT * FROM secret1;
```

VDATE	EMPNO	OLD_SAL	NEW_SAL	USER_ID
16-MAY-03	7369	800	1000	32
16-MAY-03	7499	1600	1800	32
16-MAY-03	7521	1250	1450	32
16-MAY-03	7654	1250	1450	32
16-MAY-03	7844	1500	1700	32
16-MAY-03	7876	1100	1300	32
16-MAY-03	7900	950	1150	32
16-MAY-03	7934	1300	1500	32

8 rows selected.

```
SQL> SELECT EMPNO, ENAME, SAL
2 FROM SCOTT.EMP;
```

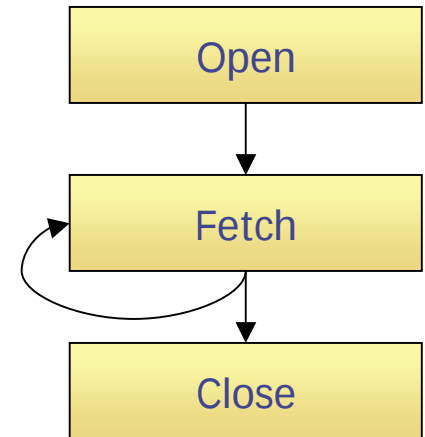
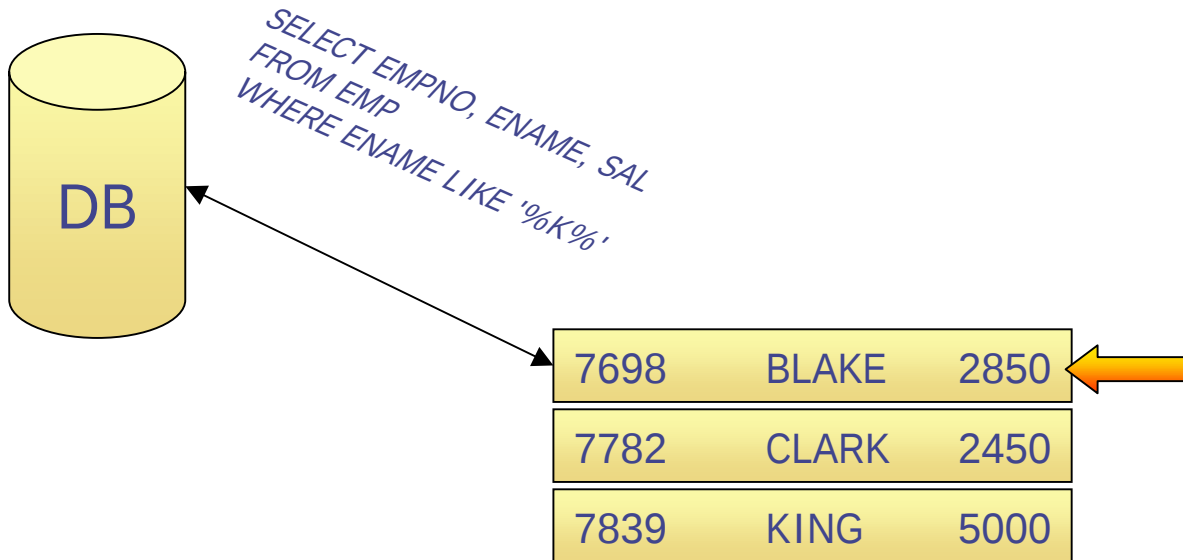
EMPNO	ENAME	SAL
7369	SMITH	1000
7499	ALLEN	1800
7521	WARD	1450
7566	JONES	3175
7654	MARTIN	1450
7698	BLAKE	3050
7782	CLARK	2650
7788	SCOTT	3200
7839	KING	5200
7844	TURNER	1700
7876	ADAMS	1300
7900	JAMES	1150
7902	FORD	3200
7934	MILLER	900

14 rows selected.

Cursors

Implicit

Explicit



Declaring

```
CURSOR cursor_name [ ( parameter [ , parameter . . . ] ) ]  
[ RETURN return specification ]  
IS  
select_statement
```

Cursors (cont.)

Opening

```
OPEN cursor_name [ ( argument[ , argument . . . ] ) ] ;
```

Fetching

```
FETCH cursor_name INTO record_or_variable list ;
```

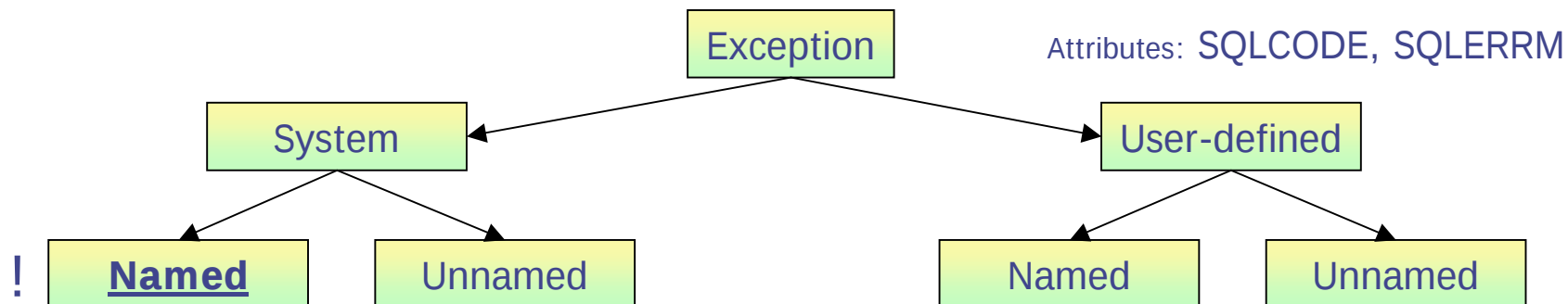
Closing

```
CLOSE cursor_name ;
```

Cursor Attributes

Name	Description
%FOUND	Returns True if record was fetched successfully
%NOTFOUND	Returns True if record was not fetched successfully
%ROWCOUNT	Returns number of records fetched from cursor at that time
%ISOPEN	Returns TRUE if cursor is open, FALSE otherwise

Exception Handling



Name of exception	SQLCODE	Description
CURSOR_ALREADY_OPEN	-6511	Attempt to open cursor that has already been opened
DUP_VAL_ON_INDEX	-1	INSERT or UPDATE statement attempts to store duplicate value in column restricted by a unique index
INVALID_CURSOR	-1001	Attempt to FETCH from or CLOSE a cursor that has not been opened yet
INVALID_NUMBER	-1722	Unsuccessful conversion a character string to a number (TO_NUMBER)
NO_DATA_FOUND	100	A SELECT INTO statement returns no rows (ORA-01403)
TOO_MANY_ROWS	-1422	A SELECT INTO statement returns more than one row
ZERO_DIVIDE	-1476	Attempt to divide by zero

LOGIN_DENIED -1017

NOT_LOGGED_ON -1012

TIMEOUT_ON_RESOURCE -51

TRANSACTION_BACKED_OUT -61

PROGRAMM_ERROR -6501

STORAGE_ERROR -6500

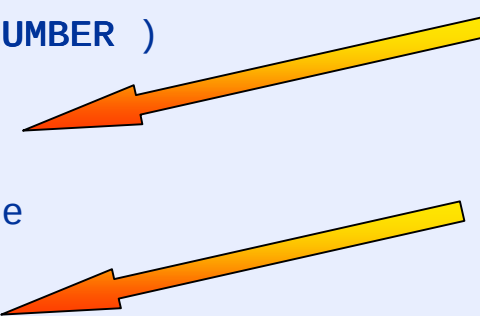
VALUE_ERROR -6502

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2   v_Num NUMBER(4) := 7499;
  3   v_Name VARCHAR2(20);
  4 BEGIN
  5   SELECT EName INTO v_Name
  6   FROM Emp
  7   WHERE EmpNo = v_Num;
  8   DBMS_OUTPUT.PUT_LINE( 'The worker ' || v_Num || ' is ' || v_Name );
  9 END;
10 /
The worker 7499 is ALLEN

PL/SQL procedure successfully completed
```

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2   v_Num NUMBER(4) := 7498;
  3   v_Name VARCHAR2(20);
  4 BEGIN
  5   SELECT EName INTO v_Name
  6   FROM Emp
  7   WHERE EmpNo = v_Num;
  8   DBMS_OUTPUT.PUT_LINE( 'The worker ' || v_Num || ' is ' || v_Name );
  9 END;
10 /
ORA-01403: no data found
ORA-06512: at line 5
```

```
SQL> CREATE OR REPLACE
2  PROCEDURE PrName( p_Num NUMBER )
3  IS
4      v_Name VARCHAR2(20);
5  BEGIN
6      SELECT EName INTO v_Name
7      FROM Emp
8      WHERE EmpNo = p_Num;
9      DBMS_OUTPUT.PUT_LINE( 'The worker ' || p_Num || ' is ' || v_Name );
10 EXCEPTION
11     WHEN NO_DATA_FOUND THEN
12         DBMS_OUTPUT.PUT_LINE( 'There is no worker ' || p_Num );
13     WHEN OTHERS THEN
14         DBMS_OUTPUT.PUT_LINE( 'Error: ' || SQLCODE || ' (' || SQLERRM || ')' );
15 END;
16 /
```



v_X NUMBER(3) := 3;
v_Y NUMBER(3) := 0;

v_Y := v_Y / v_X;

Procedure created

```
SQL> EXECUTE PrName( 7499 );
The worker 7499 is ALLEN
```

```
SQL> EXECUTE PrName( 7498 );
There is no worker 7498
```

```
SQL> EXECUTE PrName( 7499 );
Error: -1476 (ORA-01476: divisor is equal to zero)
```

```
SQL> EXECUTE PrName( 7498 );
There is no worker 7498
```

```
SQL> CREATE OR REPLACE
  2  PROCEDURE PrNumber( p_Name VARCHAR2 )
  3  IS
  4      v_Num NUMBER(4);
  5      v_Name VARCHAR2(20);
  6  BEGIN
  7      SELECT EmpNo, EName INTO v_Num, v_Name FROM emp
  8      WHERE UPPER( EName ) LIKE UPPER( p_Name );
  9      DBMS_OUTPUT.PUT_LINE( 'The worker ' || v_Name || ' has number ' || v_Num );
10  EXCEPTION
11      WHEN NO_DATA_FOUND THEN
12          DBMS_OUTPUT.PUT_LINE( 'There is no worker ' || p_Name );
13      WHEN TOO_MANY_ROWS THEN
14          DBMS_OUTPUT.PUT_LINE( 'There is more than one worker matching to '
15                                || p_Name );
16  END;
17  /
```

Procedure created

```
SQL> EXECUTE PrNumber( 'ALLEN' );
The worker ALLEN has number 7499
```

```
SQL> EXECUTE PrNumber( 'M%' );
There is more than one worker matching to M%
```

```
SQL> EXECUTE PrNumber( 'ELI' );
There is no worker ELI
```

```
CREATE OR REPLACE
PROCEDURE PrNumberM( p_Name VARCHAR2 )
IS
    v_Num NUMBER(4);
    v_Name VARCHAR2(20);
    CURSOR c1
    IS
        SELECT EmpNo, EName FROM Emp
        WHERE UPPER( EName ) LIKE UPPER( p_Name );
BEGIN
    OPEN c1;
    LOOP
        FETCH c1 INTO v_Num, v_Name;
        EXIT WHEN c1%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE( 'The worker ' || v_Name || ' has number ' || v_Num );
    END LOOP;
    IF c1%ROWCOUNT = 0 THEN
        DBMS_OUTPUT.PUT_LINE( 'There is no worker ' || p_Name );
    END IF;
END;
/
```

Procedure created

```
SQL> EXECUTE PrNumberM( 'ALLEN' );
The worker ALLEN has number 7499
```

```
SQL> EXECUTE PrNumberM( 'M%' );
The worker MARTIN has number 7654
The worker MILLER has number 7934
```

```
SQL> EXECUTE PrNumber( 'ELI' );
There is no worker ELI
```

```
CREATE OR REPLACE
PROCEDURE DEPT_DEL(p_deptno IN NUMBER)
IS
BEGIN
    DELETE FROM dept
    WHERE deptno = p_deptno;
EXCEPTION
    WHEN OTHERS THEN
        IF SQLCODE = -2292 THEN /* integrity constraint violated - child record found */
            DBMS_OUTPUT.PUT_LINE( 'It is impossible to delete this department ' ||
                                   'because of integrity violation' );
        ELSE
            RAISE;
        END IF;
    END DEPT_DEL;
```

Procedure created

```
SQL> EXECUTE DEPT_DEL( 10 );
It is impossible to delete this department because of integrity violation
```

<http://otn.oracle.com>

"Error messages"
a96525.pdf